UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/730,900	12/10/2003	Jonathan Maron	5231-087-US01	5194

68009   7590   12/31/2009

Hanify & King, P.C.
1055 Thomas Jefferson Street, NW
Suite 400
WASHINGTON, DC 20007

EXAMINER
CHEN, QING

ART UNIT	PAPER NUMBER
2191	

MAIL DATE	DELIVERY MODE
12/31/2009	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

	Application No.	Applicant(s)
	10/730,900	MARON, JONATHAN
Office Action Summary	Examiner	Art Unit	
	Qing Chen	2191	

-- *The MAILING DATE of this communication appears on the cover sheet with the correspondence address* --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE <u>3</u> MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.
- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on <u>06 October 2009</u>.

2a)☒ This action is FINAL.   2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) <u>1,4-10,13-16,19-25,28-31,34-40 and 43-51</u> is/are pending in the application.

    4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) <u>1,4-10,13-16,19-25,28-31,34-40 and 43-51</u> is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

    Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

    Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

    a)☐ All b)☐ Some * c)☐ None of:

      1.☐ Certified copies of the priority documents have been received.

      2.☐ Certified copies of the priority documents have been received in Application No. _____.

      3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

    * See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1)☐ Notice of References Cited (PTO-892)
2)☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3)☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.

4)☐ Interview Summary (PTO-413) Paper No(s)/Mail. Date. _____.
5)☐ Notice of Informal Patent Application
6)☐ Other: _____.

DETAILED ACTION

1.   This Office action is in response to the amendment filed on October 6, 2009.

2.   **Claims 1, 4-10, 13-16, 19-25, 28-31, 34-40, and 43-51** are pending.

3.   **Claims 1, 4, 13, 16, 19, 28, 31, 34, and 43** have been amended.

4.   **Claims 2, 3, 11, 12, 17, 18, 26, 27, 32, 33, 41, and 42** have been canceled.

Response to Amendment

Claim Objections

5.   **Claims 13, 28, and 43** are objected to because of the following informalities:

•   **Claim 13** contains a typographical error: Claim 13 should presumably depend on

Claim 10, not Claim 12.

•   **Claim 28** contains a typographical error: Claim 28 should presumably depend on

Claim 25, not Claim 27.

•   **Claim 43** contains a typographical error: Claim 43 should presumably depend on

Claim 40, not Claim 42.

Appropriate correction is required.

Claim Rejections - 35 USC § 103

6.   The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all

obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in
> section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are
> such that the subject matter as a whole would have been obvious at the time the invention was made to a person
> having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the
> manner in which the invention was made.

7.   **Claims 1, 4-10, 13-16, 19-25, 28-31, 34-40, and 43-51** are rejected under 35 U.S.C.

103(a) as being unpatentable over **US 6,925,631 (hereinafter "Golden")** in view of **US**

6,754,659 (hereinafter "Sarkar") and **US 2003/0158832 (hereinafter "Sijacic")**.

       As per **Claim 1,** <u>Golden</u> discloses:

       -    creating an event handler for a method node found in the markup language

description *(see Column 9: 37-41, "The SAX parser, an event-driven API, is used for the parsing*

process. The application registers an event handler to a parser object that implements the

org.sax.Parser interface. The event handler interface DocumentHandler is called whenever an

element is found in the input stream.");

       -    registering the event handler *(see Column 9: 38-40, "The application registers an*

event handler to a parser object that implements the org.sax.Parser interface.");

       -    parsing the markup language description and invoking the registered event handler

(see Column 9: 53-61, "Upon parsing the input stream, a DOM representation of it is created.

As an XML tag is found, an object (e.g. a JAVA class), as defined by the corresponding binding,

is bound to the DOM tree for the specific tag. A tag with behavior (e.g. a JAVA class) bound to it

is called a "taglet". As will be explained in more detail below, a init ( ) method is invoked on the

taglet. After all of the taglet's children (possibly zero) have been added to the DOM

representation, the taglet's run ( ) method is invoked."); and

       -    automatically generating output code using the invoked event handler *(see Column*

14: 45-46, "… the taglet document is written to the output stream 15."; Column 17: 25-38,

"FIG. 7 illustrates examples in which the above-described embodiments are used for transforming an XML input stream into a different output stream." and "The XBF engine 13 processes the XML input document 14 as described in the context of FIGS. 5 and 6, using bindings 12 which define the mapping between tags in the XML input document 14 and classes (e.g. JAVA classes), which give "behavior" to the tags. In one the examples depicted in FIG. 7, the "behavior" is the translation of the XML input document 14 into a HTML output document 50.").

However, <u>Golden</u> does not disclose:

- receiving an archive file to be deployed, wherein the archive file includes at least one input class;

- introspecting an input class included in the archive file, wherein introspecting the input class included in the archive file includes automatically generating information relating to the input class by extracting information identifying methods included in the input class; and for each method, extracting information relating to parameters of the method including at least a name and a type of each parameter; and

- automatically generating a markup language description of the input class based on the extracted information identifying methods included in the input class; and extracted information relating to parameters of the method including at least a name and a type of each parameter.

<u>Sarkar</u> discloses:

- receiving an archive file to be deployed, wherein the archive file includes at least one input class *(see Column 6: 50-55, "Then, at step 606, the Access Bean object AB701 retrieves*

environment information about generic EJB 720, locates it, and creates an instance of the

generic EJB 720 using standard EJB API calls." It is inherent that the generic EJB includes at

least one input class.); and

    -   introspecting an input class included in the archive file, wherein introspecting the

input class included in the archive file includes automatically generating information relating to

the input class by extracting information identifying methods included in the input class; and for

each method, extracting information relating to parameters of the method including at least a

name and a type of each parameter *(see Column 5: 58-64, "To create the code for the generic*

EJB, all the programmer needs to know are the classes of the support code to be generated (in

this example, described in more detail below, the classes are an "Access Bean class," the

"properties class," and the "helper class"); the programmer can then define the generic EJB to

perform reflection on these classes in a known manner."; Column 6: 66 and 67 to Column 7: 1-

4, "Specifically, at step 608, the generic EJB 720 uses Java class/introspection/reflection to

retrieve the static variable of properties object C1 and obtain the name of the helper object it

needs to instantiate which is based on the previously-set static variable." and 8-11, "At step 612,

the generic EJB 720 uses Java reflection to get the main execution method of the helper object

HO701 and, at step 614, invokes it, passing in the properties object C1 as an argument.").

     Therefore, it would have been obvious to one of ordinary skill in the art at the time the

invention was made to incorporate the teaching of <u>Sarkar</u> into the teaching of <u>Golden</u> to modify

<u>Golden</u>'s invention to include receiving an archive file to be deployed, wherein the archive file

includes at least one input class; and introspecting an input class included in the archive file,

wherein introspecting the input class included in the archive file includes automatically

generating information relating to the input class by extracting information identifying methods included in the input class; and for each method, extracting information relating to parameters of the method including at least a name and a type of each parameter. The modification would be obvious because one of ordinary skill in the art would be motivated to extract information from applications without modifying the applications themselves *(see Sarkar – Column 2: 10-13)*.

Sijacic discloses:

- automatically generating a markup language description of an input class based on extracted information identifying methods included in the input class; and extracted information relating to parameters of a method including at least a name and a type of each parameter *(see Paragraph [0119], "Once a Java class that implements the ISimpleWorkPerformer interface is created and compiled, an XML description file for the class is defined. The XML description file specifies the environment, input, and output parameters that the class uses. In addition, the XML file specifies some optional design parameters that may control the custom activity's appearance in the process builder 391."; Paragraph [0144])*. [Examiner's Remarks: Note that Table 2 in paragraph [0144] summarizes the parameter attributes of a method which include a name attribute and a type attribute.]

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Sijacic into the teaching of Golden to modify Golden's invention to include automatically generating a markup language description of the input class based on the extracted information identifying methods included in the input class; and extracted information relating to parameters of the method including at least a name and a type of each parameter. The modification would be obvious because one of ordinary skill in the

art would be motivated to describe a class of data objects using tags *(see Golden – Column 3: 40-48)*.

As per **Claim 4**, the rejection of **Claim 1** is incorporated; however, Golden does not

disclose:

-   automatically generating an Extensible Markup Language description of the input

class based on the extracted information identifying methods included in the input class; and

extracted information relating to parameters of the method including at least a name and a type

of each parameter.

Sijacic discloses:

-   automatically generating an Extensible Markup Language description of an input

class based on extracted information identifying methods included in the input class; and

extracted information relating to parameters of a method including at least a name and a type of

each parameter *(see Paragraph [0119], "Once a Java class that implements the*

ISimpleWorkPerformer interface is created and compiled, an XML description file for the class

is defined. The XML description file specifies the environment, input, and output parameters that

the class uses. In addition, the XML file specifies some optional design parameters that may

control the custom activity's appearance in the process builder 391."; Paragraph [0144]).

[Examiner's Remarks: Note that Table 2 in paragraph [0144] summarizes the parameter

attributes of a method which include a name attribute and a type attribute.]

Therefore, it would have been obvious to one of ordinary skill in the art at the time the

invention was made to incorporate the teaching of Sijacic into the teaching of Golden to modify

<u>Golden</u>'s invention to include automatically generating an Extensible Markup Language

description of the input class based on the extracted information identifying methods included in

the input class; and extracted information relating to parameters of the method including at least

a name and a type of each parameter. The modification would be obvious because one of

ordinary skill in the art would be motivated to describe a class of data objects using tags *(see*

<u>Golden</u> *– Column 3: 40-48).*

As per **Claim 5**, the rejection of **Claim 4** is incorporated; and <u>Golden</u> further discloses:

-    creating a Simple Application Programming Interface for Extensible Markup

Language event handler for a method node found in the Extensible Markup Language

description *(see Column 9: 37-41, "The SAX parser, an event-driven API, is used for the parsing*

process. The application registers an event handler to a parser object that implements the

org.sax.Parser interface. The event handler interface DocumentHandler is called whenever an

element is found in the input stream.").

As per **Claim 7**, the rejection of **Claim 1** is incorporated; and <u>Golden</u> further discloses:

-    creating a plurality of event handlers for a method node found in the markup language

description *(see Figure 2; Column 6: 51-61, "FIG. 2 illustrates that the tags of the XML*

fragment of FIG. 1 are mapped to classes of an object-oriented programming language by

arrows pointing from each tag to a class. More specifically, the arrows point to "init" methods at

the start-tags and to "run" methods at the end-tags.").

As per **Claim 8**, the rejection of **Claim 7** is incorporated; and <u>Golden</u> further discloses:

- registering each of the plurality of event handlers *(see Column 9: 37-41, "The SAX parser, an event-driven API, is used for the parsing process. The application registers an event handler to a parser object that implements the org.sax.Parser interface. The event handler interface DocumentHandler is called whenever an element is found in the input stream.")*.

As per **Claim 9**, the rejection of **Claim 8** is incorporated; and <u>Golden</u> further discloses:

- parsing the markup language description and invoking each of the plurality of registered event handlers *(see Column 9: 53-61, "Upon parsing the input stream, a DOM representation of it is created. As an XML tag is found, an object (e.g. a JAVA class), as defined by the corresponding binding, is bound to the DOM tree for the specific tag. A tag with behavior (e.g. a JAVA class) bound to it is called a "taglet". As will be explained in more detail below, a init ( ) method is invoked on the taglet. After all of the taglet's children (possibly zero) have been added to the DOM representation, the taglet's run ( ) method is invoked.")*.

As per **Claim 10**, the rejection of **Claim 9** is incorporated; and <u>Golden</u> further discloses:

- automatically generating output code using each of the plurality of invoked event handler in parallel *(see Column 14: 45-46, "... the taglet document is written to the output stream 15."; Column 18: 12-13, "If the branching is not conditional, the two branches following the first engine work in parallel.")*.

As per **Claim 13**, the rejection of **Claim 10** is incorporated; however, <u>Golden</u> does not disclose:

-       automatically generating an Extensible Markup Language description of the input class based on the extracted information identifying methods included in the input class; and extracted information relating to parameters of the method including at least a name and a type of each parameter.

<u>Sijacic</u> discloses:

-       automatically generating an Extensible Markup Language description of an input class based on extracted information identifying methods included in the input class; and extracted information relating to parameters of a method including at least a name and a type of each parameter *(see Paragraph [0119], "Once a Java class that implements the ISimpleWorkPerformer interface is created and compiled, an XML description file for the class is defined. The XML description file specifies the environment, input, and output parameters that the class uses. In addition, the XML file specifies some optional design parameters that may control the custom activity's appearance in the process builder 391."; Paragraph [0144]).* [Examiner's Remarks: Note that Table 2 in paragraph [0144] summarizes the parameter attributes of a method which include a name attribute and a type attribute.]

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of <u>Sijacic</u> into the teaching of <u>Golden</u> to modify <u>Golden</u>'s invention to include automatically generating an Extensible Markup Language description of the input class based on the extracted information identifying methods included in the input class; and extracted information relating to parameters of the method including at least

a name and a type of each parameter. The modification would be obvious because one of

ordinary skill in the art would be motivated to describe a class of data objects using tags *(see*

Golden – Column 3: 40-48).

As per **Claim 14**, the rejection of **Claim 13** is incorporated; and <u>Golden</u> further discloses:

-　creating a plurality of Simple Application Programming Interface for Extensible

Markup Language event handlers for a method node found in the Extensible Markup Language

description *(see Figure 2; Column 6: 51-61, "FIG. 2 illustrates that the tags of the XML*

fragment of FIG. 1 are mapped to classes of an object-oriented programming language by

arrows pointing from each tag to a class. More specifically, the arrows point to "init" methods at

the start-tags and to "run" methods at the end-tags.").

As per **Claim 16**, <u>Golden</u> discloses:

-　a processor operable to execute computer program instructions *(see Column 7: 34-35,*

"... a computer system 10 with a processing unit and storage 11 for processing programs.");

-　a memory operable to store computer program instructions executable by the

processor *(see Column 7: 34-35, "... a computer system 10 with a processing unit and storage*

11 for processing programs."); and

-　computer program instructions stored in the memory and executable *(see Column 6:*

25-28, "The disclosed embodiments of the computer program product comprise the disclosed

program code which, for example, is stored on a computer-readable data carrier ...") to perform

the steps of:

- creating an event handler for a method node found in the markup language

description *(see Column 9: 37-41, "The SAX parser, an event-driven API, is used for the parsing*

process. The application registers an event handler to a parser object that implements the

org.sax.Parser interface. The event handler interface DocumentHandler is called whenever an

element is found in the input stream.");

- registering the event handler *(see Column 9: 38-40, "The application registers an*

event handler to a parser object that implements the org.sax.Parser interface.");

- parsing the markup language description and invoking the registered event handler

(see Column 9: 53-61, "Upon parsing the input stream, a DOM representation of it is created.

As an XML tag is found, an object (e.g. a JAVA class), as defined by the corresponding binding,

is bound to the DOM tree for the specific tag. A tag with behavior (e.g. a JAVA class) bound to it

is called a "taglet". As will be explained in more detail below, a init ( ) method is invoked on the

taglet. After all of the taglet's children (possibly zero) have been added to the DOM

representation, the taglet's run ( ) method is invoked."); and

- automatically generating output code using the invoked event handler *(see Column*

14: 45-46, "... the taglet document is written to the output stream 15."; Column 17: 25-38,

"FIG. 7 illustrates examples in which the above-described embodiments are used for

transforming an XML input stream into a different output stream." and "The XBF engine 13

processes the XML input document 14 as described in the context of FIGS. 5 and 6, using

bindings 12 which define the mapping between tags in the XML input document 14 and classes

(e.g. JAVA classes), which give "behavior" to the tags. In one the examples depicted in FIG. 7,

the "behavior" is the translation of the XML input document 14 into a HTML output document 50.").

However, <u>Golden</u> does not disclose:

- receiving an archive file to be deployed, wherein the archive file includes at least one input class;

- introspecting an input class included in the archive file, wherein introspecting the input class included in the archive file includes automatically generating information relating to the input class by extracting information identifying methods included in the input class; and for each method, extracting information relating to parameters of the method including at least a name and a type of each parameter; and

- automatically generating a markup language description of the input class based on the extracted information identifying methods included in the input class; and extracted information relating to parameters of the method including at least a name and a type of each parameter.

<u>Sarkar</u> discloses:

- receiving an archive file to be deployed, wherein the archive file includes at least one input class *(see Column 6: 50-55, "Then, at step 606, the Access Bean object AB701 retrieves environment information about generic EJB 720, locates it, and creates an instance of the generic EJB 720 using standard EJB API calls." It is inherent that the generic EJB includes at least one input class.)*; and

- introspecting an input class included in the archive file, wherein introspecting the input class included in the archive file includes automatically generating information relating to

the input class by extracting information identifying methods included in the input class; and for

each method, extracting information relating to parameters of the method including at least a

name and a type of each parameter *(see Column 5: 58-64, "To create the code for the generic*

EJB, all the programmer needs to know are the classes of the support code to be generated (in

this example, described in more detail below, the classes are an "Access Bean class," the

"properties class," and the "helper class"); the programmer can then define the generic EJB to

perform reflection on these classes in a known manner."; Column 6: 66 and 67 to Column 7: 1-

4, "Specifically, at step 608, the generic EJB 720 uses Java class/introspection/reflection to

retrieve the static variable of properties object C1 and obtain the name of the helper object it

needs to instantiate which is based on the previously-set static variable." and 8-11, "At step 612,

the generic EJB 720 uses Java reflection to get the main execution method of the helper object

HO701 and, at step 614, invokes it, passing in the properties object C1 as an argument.").

Therefore, it would have been obvious to one of ordinary skill in the art at the time the

invention was made to incorporate the teaching of Sarkar into the teaching of Golden to modify

Golden's invention to include receiving an archive file to be deployed, wherein the archive file

includes at least one input class; and introspecting an input class included in the archive file,

wherein introspecting the input class included in the archive file includes automatically

generating information relating to the input class by extracting information identifying methods

included in the input class; and for each method, extracting information relating to parameters of

the method including at least a name and a type of each parameter. The modification would be

obvious because one of ordinary skill in the art would be motivated to extract information from

applications without modifying the applications themselves *(see Sarkar – Column 2: 10-13).*

Sijacic discloses:

-    automatically generating a markup language description of an input class based on

extracted information identifying methods included in the input class; and extracted information

relating to parameters of a method including at least a name and a type of each parameter *(see*

Paragraph [0119], "Once a Java class that implements the ISimpleWorkPerformer interface is

created and compiled, an XML description file for the class is defined. The XML description file

specifies the environment, input, and output parameters that the class uses. In addition, the XML

file specifies some optional design parameters that may control the custom activity's appearance

in the process builder 391.").

Therefore, it would have been obvious to one of ordinary skill in the art at the time the

invention was made to incorporate the teaching of Sijacic into the teaching of Golden to modify

Golden's invention to include automatically generating a markup language description of the

input class based on the extracted information identifying methods included in the input class;

and extracted information relating to parameters of the method including at least a name and a

type of each parameter. The modification would be obvious because one of ordinary skill in the

art would be motivated to describe a class of data objects using tags *(see Golden – Column 3: 40-*

48).

As per **Claim 19**, the rejection of **Claim 16** is incorporated; however, Golden does not

disclose:

-    automatically generating an Extensible Markup Language description of the input

class based on the extracted information identifying methods included in the input class; and

extracted information relating to parameters of the method including at least a name and a type
of each parameter.

Sijacic discloses:

-   automatically generating an Extensible Markup Language description of an input
class based on extracted information identifying methods included in the input class; and
extracted information relating to parameters of a method including at least a name and a type of
each parameter *(see Paragraph [0119], "Once a Java class that implements the
ISimpleWorkPerformer interface is created and compiled, an XML description file for the class
is defined. The XML description file specifies the environment, input, and output parameters that
the class uses. In addition, the XML file specifies some optional design parameters that may
control the custom activity's appearance in the process builder 391."; Paragraph [0144]).*
[Examiner's Remarks: Note that Table 2 in paragraph [0144] summarizes the parameter
attributes of a method which include a name attribute and a type attribute.]

Therefore, it would have been obvious to one of ordinary skill in the art at the time the
invention was made to incorporate the teaching of Sijacic into the teaching of Golden to modify
Golden's invention to include automatically generating an Extensible Markup Language
description of the input class based on the extracted information identifying methods included in
the input class; and extracted information relating to parameters of the method including at least
a name and a type of each parameter. The modification would be obvious because one of
ordinary skill in the art would be motivated to describe a class of data objects using tags *(see
Golden – Column 3: 40-48).*

As per **Claim 20**, the rejection of **Claim 19** is incorporated; and <u>Golden</u> further discloses:

- creating a Simple Application Programming Interface for Extensible Markup Language event handler for a method node found in the Extensible Markup Language description *(see Column 9: 37-41, "The SAX parser, an event-driven API, is used for the parsing process. The application registers an event handler to a parser object that implements the org.sax.Parser interface. The event handler interface DocumentHandler is called whenever an element is found in the input stream.").*

As per **Claim 31**, <u>Golden</u> discloses:

- a computer readable medium *(see Column 6: 25-28, "... a computer-readable data carrier ...")*; and

- computer program instructions, recorded on the computer readable medium, executable by a processor, *(see Column 6: 25-28, "The disclosed embodiments of the computer program product comprise the disclosed program code which, for example, is stored on a computer-readable data carrier ...")* for performing the steps of:

- creating an event handler for a method node found in the markup language description *(see Column 9: 37-41, "The SAX parser, an event-driven API, is used for the parsing process. The application registers an event handler to a parser object that implements the org.sax.Parser interface. The event handler interface DocumentHandler is called whenever an element is found in the input stream.")*;

- registering the event handler *(see Column 9: 38-40, "The application registers an event handler to a parser object that implements the org.sax.Parser interface.")*;

- parsing the markup language description and invoking the registered event handler

(see Column 9: 53-61, "Upon parsing the input stream, a DOM representation of it is created.

As an XML tag is found, an object (e.g. a JAVA class), as defined by the corresponding binding,

is bound to the DOM tree for the specific tag. A tag with behavior (e.g. a JAVA class) bound to it

is called a "taglet". As will be explained in more detail below, a init ( ) method is invoked on the

taglet. After all of the taglet's children (possibly zero) have been added to the DOM

representation, the taglet's run ( ) method is invoked."); and

- automatically generating output code using the invoked event handler *(see Column*

14: 45-46, "... the taglet document is written to the output stream 15."; Column 17: 25-38,

"FIG. 7 illustrates examples in which the above-described embodiments are used for

transforming an XML input stream into a different output stream." and "The XBF engine 13

processes the XML input document 14 as described in the context of FIGS. 5 and 6, using

bindings 12 which define the mapping between tags in the XML input document 14 and classes

(e.g. JAVA classes), which give "behavior" to the tags. In one the examples depicted in FIG. 7,

the "behavior" is the translation of the XML input document 14 into a HTML output document

50.").

However, <u>Golden</u> does not disclose:

- receiving an archive file to be deployed, wherein the archive file includes at least one

input class;

- introspecting an input class included in the archive file, wherein introspecting the

input class included in the archive file includes automatically generating information relating to

the input class by extracting information identifying methods included in the input class; and for

each method, extracting information relating to parameters of the method including at least a

name and a type of each parameter; and

> - automatically generating a markup language description of the input class based on

the extracted information identifying methods included in the input class; and extracted

information relating to parameters of the method including at least a name and a type of each

parameter.

> Sarkar discloses:

> - receiving an archive file to be deployed, wherein the archive file includes at least one

input class *(see Column 6: 50-55, "Then, at step 606, the Access Bean object AB701 retrieves*

environment information about generic EJB 720, locates it, and creates an instance of the

generic EJB 720 using standard EJB API calls." It is inherent that the generic EJB includes at

least one input class.); and

> - introspecting an input class included in the archive file, wherein introspecting the

input class included in the archive file includes automatically generating information relating to

the input class by extracting information identifying methods included in the input class; and for

each method, extracting information relating to parameters of the method including at least a

name and a type of each parameter *(see Column 5: 58-64, "To create the code for the generic*

EJB, all the programmer needs to know are the classes of the support code to be generated (in

this example, described in more detail below, the classes are an "Access Bean class," the

"properties class," and the "helper class"); the programmer can then define the generic EJB to

perform reflection on these classes in a known manner."; Column 6: 66 and 67 to Column 7: 1-

4, "Specifically, at step 608, the generic EJB 720 uses Java class/introspection/reflection to

retrieve the static variable of properties object C1 and obtain the name of the helper object it

needs to instantiate which is based on the previously-set static variable." and 8-11, "At step 612,

the generic EJB 720 uses Java reflection to get the main execution method of the helper object

HO701 and, at step 614, invokes it, passing in the properties object C1 as an argument.").

Therefore, it would have been obvious to one of ordinary skill in the art at the time the

invention was made to incorporate the teaching of <u>Sarkar</u> into the teaching of <u>Golden</u> to modify

<u>Golden</u>'s invention to include receiving an archive file to be deployed, wherein the archive file

includes at least one input class; and introspecting an input class included in the archive file,

wherein introspecting the input class included in the archive file includes automatically

generating information relating to the input class by extracting information identifying methods

included in the input class; and for each method, extracting information relating to parameters of

the method including at least a name and a type of each parameter. The modification would be

obvious because one of ordinary skill in the art would be motivated to extract information from

applications without modifying the applications themselves *(see <u>Sarkar</u> – Column 2: 10-13).*

<u>Sijacic</u> discloses:

-    automatically generating a markup language description of an input class based on

extracted information identifying methods included in the input class; and extracted information

relating to parameters of a method including at least a name and a type of each parameter *(see*

Paragraph [0119], "Once a Java class that implements the ISimpleWorkPerformer interface is

created and compiled, an XML description file for the class is defined. The XML description file

specifies the environment, input, and output parameters that the class uses. In addition, the XML

file specifies some optional design parameters that may control the custom activity's appearance

in the process builder 391."; *Paragraph [0144]).* [Examiner's Remarks: Note that Table 2 in

paragraph [0144] summarizes the parameter attributes of a method which include a name

attribute and a type attribute.]

Therefore, it would have been obvious to one of ordinary skill in the art at the time the

invention was made to incorporate the teaching of Sijacic into the teaching of Golden to modify

Golden's invention to include automatically generating a markup language description of the

input class based on the extracted information identifying methods included in the input class;

and extracted information relating to parameters of the method including at least a name and a

type of each parameter. The modification would be obvious because one of ordinary skill in the

art would be motivated to describe a class of data objects using tags *(see Golden – Column 3: 40-*

48).

As per **Claim 34**, the rejection of **Claim 31** is incorporated; however, Golden does not

disclose:

-    automatically generating an Extensible Markup Language description of the input

class based on the extracted information identifying methods included in the input class; and

extracted information relating to parameters of the method including at least a name and a type

of each parameter.

Sijacic discloses:

-    automatically generating an Extensible Markup Language description of an input

class based on extracted information identifying methods included in the input class; and

extracted information relating to parameters of a method including at least a name and a type of

each parameter *(see Paragraph [0119], "Once a Java class that implements the*

ISimpleWorkPerformer interface is created and compiled, an XML description file for the class

is defined. The XML description file specifies the environment, input, and output parameters that

the class uses. In addition, the XML file specifies some optional design parameters that may

control the custom activity's appearance in the process builder 391."; Paragraph [0144]).

[Examiner's Remarks: Note that Table 2 in paragraph [0144] summarizes the parameter

attributes of a method which include a name attribute and a type attribute.]

Therefore, it would have been obvious to one of ordinary skill in the art at the time the

invention was made to incorporate the teaching of Sijacic into the teaching of Golden to modify

Golden's invention to include automatically generating an Extensible Markup Language

description of the input class based on the extracted information identifying methods included in

the input class; and extracted information relating to parameters of the method including at least

a name and a type of each parameter. The modification would be obvious because one of

ordinary skill in the art would be motivated to describe a class of data objects using tags *(see*

Golden – Column 3: 40-48).

As per **Claim 35**, the rejection of **Claim 34** is incorporated; and Golden further discloses:

-   creating a Simple Application Programming Interface for Extensible Markup

Language event handler for a method node found in the Extensible Markup Language

description *(see Column 9: 37-41, "The SAX parser, an event-driven API, is used for the parsing*

process. The application registers an event handler to a parser object that implements the

org.sax.Parser interface. The event handler interface DocumentHandler is called whenever an

element is found in the input stream.").

As per **Claim 37**, the rejection of **Claim 31** is incorporated; and <u>Golden</u> further discloses:

-    creating a plurality of event handlers for a method node found in the markup language

description *(see Figure 2; Column 6: 51-61, "FIG. 2 illustrates that the tags of the XML*

fragment of FIG. 1 are mapped to classes of an object-oriented programming language by

arrows pointing from each tag to a class. More specifically, the arrows point to "init" methods at

the start-tags and to "run" methods at the end-tags.").

As per **Claim 38**, the rejection of **Claim 37** is incorporated; and <u>Golden</u> further discloses:

-    registering each of the plurality of event handlers *(see Column 9: 37-41, "The SAX*

parser, an event-driven API, is used for the parsing process. The application registers an event

handler to a parser object that implements the org.sax.Parser interface. The event handler

interface DocumentHandler is called whenever an element is found in the input stream.").

As per **Claim 39**, the rejection of **Claim 38** is incorporated; and <u>Golden</u> further discloses:

-    parsing the markup language description and invoking each of the plurality of

registered event handlers *(see Column 9: 53-61, "Upon parsing the input stream, a DOM*

representation of it is created. As an XML tag is found, an object (e.g. a JAVA class), as defined

by the corresponding binding, is bound to the DOM tree for the specific tag. A tag with behavior

(e.g. a JAVA class) bound to it is called a "taglet". As will be explained in more detail below, a

init ( ) method is invoked on the taglet. After all of the taglet's children (possibly zero) have been

added to the DOM representation, the taglet's run ( ) method is invoked.").

As per **Claim 40**, the rejection of **Claim 39** is incorporated; and <u>Golden</u> further discloses:

- automatically generating output code using each of the plurality of invoked event

handler in parallel *(see Column 14: 45-46, "... the taglet document is written to the output*

stream 15."; Column 18: 12-13, "If the branching is not conditional, the two branches following

the first engine work in parallel.").

As per **Claim 43**, the rejection of **Claim 40** is incorporated; however, <u>Golden</u> does not

disclose:

- automatically generating an Extensible Markup Language description of the input

class based on the extracted information identifying methods included in the input class; and

extracted information relating to parameters of the method including at least a name and a type

of each parameter.

<u>Sijacic</u> discloses:

- automatically generating an Extensible Markup Language description of an input

class based on extracted information identifying methods included in the input class; and

extracted information relating to parameters of a method including at least a name and a type of

each parameter *(see Paragraph [0119], "Once a Java class that implements the*

ISimpleWorkPerformer interface is created and compiled, an XML description file for the class

is defined. The XML description file specifies the environment, input, and output parameters that

the class uses. In addition, the XML file specifies some optional design parameters that may

control the custom activity's appearance in the process builder 391."; Paragraph [0144]).

[Examiner's Remarks: Note that Table 2 in paragraph [0144] summarizes the parameter

attributes of a method which include a name attribute and a type attribute.]

     Therefore, it would have been obvious to one of ordinary skill in the art at the time the

invention was made to incorporate the teaching of <u>Sijacic</u> into the teaching of <u>Golden</u> to modify

<u>Golden</u>'s invention to include automatically generating an Extensible Markup Language

description of the input class based on the extracted information identifying methods included in

the input class; and extracted information relating to parameters of the method including at least

a name and a type of each parameter. The modification would be obvious because one of

ordinary skill in the art would be motivated to describe a class of data objects using tags *(see*

<u>Golden</u> – Column 3: 40-48).

     As per **Claim 44**, the rejection of **Claim 43** is incorporated; and <u>Golden</u> further discloses:

-   creating a plurality of Simple Application Programming Interface for Extensible

Markup Language event handlers for a method node found in the Extensible Markup Language

description *(see Figure 2; Column 6: 51-61, "FIG. 2 illustrates that the tags of the XML*

fragment of FIG. 1 are mapped to classes of an object-oriented programming language by

arrows pointing from each tag to a class. More specifically, the arrows point to "init" methods at

the start-tags and to "run" methods at the end-tags.").

     As per **Claim 46**, the rejection of **Claim 5** is incorporated; and <u>Golden</u> further discloses:

registering the created Simple Application Programming Interface for Extensible Markup Language event handler for a method node found in the Extensible Markup Language description *(see Column 9: 37-41, "The SAX parser, an event-driven API, is used for the parsing process. The application registers an event handler to a parser object that implements the org.sax.Parser interface. The event handler interface DocumentHandler is called whenever an element is found in the input stream.").*

As per **Claim 6**, the rejection of **Claim 46** is incorporated; and <u>Golden</u> further discloses:

- parsing the Extensible Markup Language description using a Simple Application Programming Interface for Extensible Markup Language parser and invoking the Simple Application Programming Interface for Extensible Markup Language event handler *(see Column 9: 53-61, "Upon parsing the input stream, a DOM representation of it is created. As an XML tag is found, an object (e.g. a JAVA class), as defined by the corresponding binding, is bound to the DOM tree for the specific tag. A tag with behavior (e.g. a JAVA class) bound to it is called a "taglet". As will be explained in more detail below, a init ( ) method is invoked on the taglet. After all of the taglet's children (possibly zero) have been added to the DOM representation, the taglet's run ( ) method is invoked.").*

As per **Claim 47**, the rejection of **Claim 20** is incorporated; and <u>Golden</u> further discloses:

- registering the created Simple Application Programming Interface for Extensible Markup Language event handler for a method node found in the Extensible Markup Language description *(see Column 9: 37-41, "The SAX parser, an event-driven API, is used for the parsing*

process. The application registers an event handler to a parser object that implements the

org.sax.Parser interface. The event handler interface DocumentHandler is called whenever an

element is found in the input stream.").

        As per **Claim 21,** the rejection of **Claim 47** is incorporated; and <u>Golden</u> further discloses:

-    parsing the Extensible Markup Language description using a Simple Application

Programming Interface for Extensible Markup Language parser and invoking the Simple

Application Programming Interface for Extensible Markup Language event handler *(see Column*

9: 53-61, "Upon parsing the input stream, a DOM representation of it is created. As an XML tag

is found, an object (e.g. a JAVA class), as defined by the corresponding binding, is bound to the

DOM tree for the specific tag. A tag with behavior (e.g. a JAVA class) bound to it is called a

"taglet". As will be explained in more detail below, a init ( ) method is invoked on the taglet.

After all of the taglet's children (possibly zero) have been added to the DOM representation, the

taglet's run ( ) method is invoked.").

        As per **Claim 22,** the rejection of **Claim 21** is incorporated; and <u>Golden</u> further discloses:

-    creating a plurality of event handlers for a method node found in the Extensible

Markup Language description *(see Figure 2; Column 6: 51-61, "FIG. 2 illustrates that the tags*

of the XML fragment of FIG. 1 are mapped to classes of an object-oriented programming

language by arrows pointing from each tag to a class. More specifically, the arrows point to

"init" methods at the start-tags and to "run" methods at the end-tags.").

As per **Claim 23**, the rejection of **Claim 22** is incorporated; and <u>Golden</u> further discloses:

- registering each of the plurality of event handlers *(see Column 9: 37-41, "The SAX parser, an event-driven API, is used for the parsing process. The application registers an event handler to a parser object that implements the org.sax.Parser interface. The event handler interface DocumentHandler is called whenever an element is found in the input stream.")*.

As per **Claim 24**, the rejection of **Claim 23** is incorporated; and <u>Golden</u> further discloses:

- parsing the Extensible Markup Language description and invoking each of the plurality of registered event handlers *(see Column 9: 53-61, "Upon parsing the input stream, a DOM representation of it is created. As an XML tag is found, an object (e.g. a JAVA class), as defined by the corresponding binding, is bound to the DOM tree for the specific tag. A tag with behavior (e.g. a JAVA class) bound to it is called a "taglet". As will be explained in more detail below, a init ( ) method is invoked on the taglet. After all of the taglet's children (possibly zero) have been added to the DOM representation, the taglet's run ( ) method is invoked.")*.

As per **Claim 25**, the rejection of **Claim 24** is incorporated; and <u>Golden</u> further discloses:

- automatically generating output code using each of the plurality of invoked event handler in parallel *(see Column 14: 45-46, "... the taglet document is written to the output stream 15."; Column 18: 12-13, "If the branching is not conditional, the two branches following the first engine work in parallel.")*.

As per **Claim 28**, the rejection of **Claim 25** is incorporated; however, <u>Golden</u> does not disclose:

-   automatically generating an Extensible Markup Language description of the input class based on the extracted information identifying methods included in the input class; and extracted information relating to parameters of the method including at least a name and a type of each parameter.

<u>Sijacic</u> discloses:

-   automatically generating an Extensible Markup Language description of an input class based on extracted information identifying methods included in the input class; and extracted information relating to parameters of a method including at least a name and a type of each parameter *(see Paragraph [0119], "Once a Java class that implements the ISimpleWorkPerformer interface is created and compiled, an XML description file for the class is defined. The XML description file specifies the environment, input, and output parameters that the class uses. In addition, the XML file specifies some optional design parameters that may control the custom activity's appearance in the process builder 391."; Paragraph [0144]).* [Examiner's Remarks: Note that Table 2 in paragraph [0144] summarizes the parameter attributes of a method which include a name attribute and a type attribute.]

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of <u>Sijacic</u> into the teaching of <u>Golden</u> to modify <u>Golden</u>'s invention to include automatically generating an Extensible Markup Language description of the input class based on the extracted information identifying methods included in the input class; and extracted information relating to parameters of the method including at least

a name and a type of each parameter. The modification would be obvious because one of

ordinary skill in the art would be motivated to describe a class of data objects using tags *(see*

Golden – *Column 3: 40-48).*

As per **Claim 29**, the rejection of **Claim 28** is incorporated; and <u>Golden</u> further discloses:

- creating a plurality of Simple Application Programming Interface for Extensible

Markup Language event handlers for a method node found in the Extensible Markup Language

description *(see Figure 2; Column 6: 51-61, "FIG. 2 illustrates that the tags of the XML*

fragment of FIG. 1 are mapped to classes of an object-oriented programming language by

arrows pointing from each tag to a class. More specifically, the arrows point to "init" methods at

the start-tags and to "run" methods at the end-tags.").

As per **Claim 48**, the rejection of **Claim 35** is incorporated; and <u>Golden</u> further discloses:

- registering the created Simple Application Programming Interface for Extensible

Markup Language event handler for a method node found in the Extensible Markup Language

description *(see Column 9: 37-41, "The SAX parser, an event-driven API, is used for the parsing*

process. The application registers an event handler to a parser object that implements the

org.sax.Parser interface. The event handler interface DocumentHandler is called whenever an

element is found in the input stream.").

As per **Claim 36**, the rejection of **Claim 48** is incorporated; and <u>Golden</u> further discloses:

- parsing the Extensible Markup Language description using a Simple Application

Programming Interface for Extensible Markup Language parser and invoking the Simple

Application Programming Interface for Extensible Markup Language event handler *(see Column*

9: 53-61, "Upon parsing the input stream, a DOM representation of it is created. As an XML tag

is found, an object (e.g. a JAVA class), as defined by the corresponding binding, is bound to the

DOM tree for the specific tag. A tag with behavior (e.g. a JAVA class) bound to it is called a

"taglet". As will be explained in more detail below, a init ( ) method is invoked on the taglet.

After all of the taglet's children (possibly zero) have been added to the DOM representation, the

taglet's run ( ) method is invoked.").

As per **Claim 49**, the rejection of **Claim 44** is incorporated; and <u>Golden</u> further discloses:

- registering the plurality of created Simple Application Programming Interface for

Extensible Markup Language event handlers for a method node found in the Extensible Markup

Language description *(see Column 9: 37-41, "The SAX parser, an event-driven API, is used for*

the parsing process. The application registers an event handler to a parser object that

implements the org.sax.Parser interface. The event handler interface DocumentHandler is called

whenever an element is found in the input stream.").

As per **Claim 45**, the rejection of **Claim 49** is incorporated; and <u>Golden</u> further discloses:

- parsing the Extensible Markup Language description using a Simple Application

Programming Interface for Extensible Markup Language parser and invoking the plurality of

Simple Application Programming Interface for Extensible Markup Language event handlers *(see*

Column 9: 53-61, "Upon parsing the input stream, a DOM representation of it is created. As an

XML tag is found, an object (e.g. a JAVA class), as defined by the corresponding binding, is

bound to the DOM tree for the specific tag. A tag with behavior (e.g. a JAVA class) bound to it is

called a "taglet". As will be explained in more detail below, a init ( ) method is invoked on the

taglet. After all of the taglet's children (possibly zero) have been added to the DOM

representation, the taglet's run ( ) method is invoked.").

As per **Claim 50**, the rejection of **Claim 14** is incorporated; and <u>Golden</u> further discloses:

-    registering the plurality of created Simple Application Programming Interface for

Extensible Markup Language event handlers for a method node found in the Extensible Markup

Language description *(see Column 9: 37-41, "The SAX parser, an event-driven API, is used for*

the parsing process. The application registers an event handler to a parser object that

implements the org.sax.Parser interface. The event handler interface DocumentHandler is called

whenever an element is found in the input stream.").

As per **Claim 15**, the rejection of **Claim 50** is incorporated; and <u>Golden</u> further discloses:

-    parsing the Extensible Markup Language description using a Simple Application

Programming Interface for Extensible Markup Language parser and invoking the plurality of

Simple Application Programming Interface for Extensible Markup Language event handlers *(see*

Column 9: 53-61, "Upon parsing the input stream, a DOM representation of it is created. As an

XML tag is found, an object (e.g. a JAVA class), as defined by the corresponding binding, is

bound to the DOM tree for the specific tag. A tag with behavior (e.g. a JAVA class) bound to it is

called a "taglet". As will be explained in more detail below, a init ( ) method is invoked on the

taglet. After all of the taglet's children (possibly zero) have been added to the DOM

representation, the taglet's run ( ) method is invoked.").

As per **Claim 51**, the rejection of **Claim 29** is incorporated; and <u>Golden</u> further discloses:

-    registering the plurality of created Simple Application Programming Interface for

Extensible Markup Language event handlers for a method node found in the Extensible Markup

Language description *(see Column 9: 37-41, "The SAX parser, an event-driven API, is used for*

the parsing process. The application registers an event handler to a parser object that

implements the org.sax.Parser interface. The event handler interface DocumentHandler is called

whenever an element is found in the input stream.").

As per **Claim 30**, the rejection of **Claim 51** is incorporated; and <u>Golden</u> further discloses:

-    parsing the Extensible Markup Language description using a Simple Application

Programming Interface for Extensible Markup Language parser and invoking the plurality of

Simple Application Programming Interface for Extensible Markup Language event handlers *(see*

Column 9: 53-61, "Upon parsing the input stream, a DOM representation of it is created. As an

XML tag is found, an object (e.g. a JAVA class), as defined by the corresponding binding, is

bound to the DOM tree for the specific tag. A tag with behavior (e.g. a JAVA class) bound to it is

called a "taglet". As will be explained in more detail below, a init ( ) method is invoked on the

taglet. After all of the taglet's children (possibly zero) have been added to the DOM

representation, the taglet's run ( ) method is invoked.").

Response to Arguments

8.   Applicant's arguments filed on October 6, 2009 have been fully considered, but they are

not persuasive.

In the Remarks, Applicant argues:

a)   For example, claims 1, 16, and 31 require introspecting an input class included in the

archive file, wherein introspecting the input class included in the archive file includes

automatically generating information relating to the input class by extracting information

identifying methods included in the input class; and for each method, extracting information

relating to parameters of the method including at least a name and a type of each parameter and

automatically generating a markup language description of the input class based on the extracted

information identifying methods included in the input class; and extracted information relating to

parameters of the method including at least a name and a type of each parameter. As the

Examiner indicates, Golden does not disclose these requirements. The Examiner cites Sarkar at

col. 6, line 66 to col. 7, line 4 as disclosing wherein introspecting the input class included in the

archive file includes automatically generating information relating to the input class by

extracting information identifying methods included in the input class; and for each method,

extracting information relating to parameters of the method including at least a name and a type

of each parameter. However, the combination of Golden and Sarkar fails to disclose the

limitation automatically generating a markup language description of the input class based on the

extracted information identifying methods included in the input class; and extracted information

relating to parameters of the method including at least a name and a type of each parameter.

However, there is no disclosure is Sijacic that any markup language description that is generated

is based on extracted information identifying methods and parameters of methods of an input

class.

Examiner's response:

a)   Examiner disagrees. With respect to the Applicant's assertion that there is no disclosure

is Sijacic that any markup language description that is generated is based on extracted

information identifying methods and parameters of methods of an input class, the Examiner

respectfully submits that Sijacic clearly discloses "automatically generating a markup language

description of an input class based on extracted information identifying methods included in the

input class; and extracted information relating to parameters of a method including at least a

name and a type of each parameter" *(see Paragraph [0119], "Once a Java class that implements*

the ISimpleWorkPerformer interface is created and compiled, an XML description file for the

class is defined. The XML description file specifies the environment, input, and output

parameters that the class uses. In addition, the XML file specifies some optional design

parameters that may control the custom activity's appearance in the process builder 391.";

Paragraph [0144]). Note that Table 2 in paragraph [0144] summarizes the parameter attributes

of a method which include a name attribute and a type attribute.

        Therefore, for at least the reason set forth above, the rejections made under 35 U.S.C. §

103(a) with respect to Claims 1, 16, and 31 are proper and therefore, maintained.

Conclusion

9.   Applicant's amendment necessitated the new ground(s) of rejection presented in this

Office action. Accordingly, **THIS ACTION IS MADE FINAL**. See MPEP § 706.07(a).

Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE

MONTHS from the mailing date of this action. In the event a first reply is filed within TWO

MONTHS of the mailing date of this final action and the advisory action is not mailed until after

the end of the THREE-MONTH shortened statutory period, then the shortened statutory period

will expire on the date the advisory action is mailed, and any extension fee pursuant to 37

CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event,

however, will the statutory period for reply expire later than SIX MONTHS from the date of this

final action.

10.   Any inquiry concerning this communication or earlier communications from the

Examiner should be directed to Qing Chen whose telephone number is 571-270-1071. The

Examiner can normally be reached on Monday through Thursday from 7:30 AM to 4:00 PM.

The Examiner can also be reached on alternate Fridays.

If attempts to reach the Examiner by telephone are unsuccessful, the Examiner's

supervisor, Wei Zhen, can be reached on 571-272-3708. The fax phone number for the

organization where this application or proceeding is assigned is 571-273-8300.

Any inquiry of a general nature or relating to the status of this application or proceeding

should be directed to the TC 2100 Group receptionist whose telephone number is 571-272-2100.

Information regarding the status of an application may be obtained from the Patent

Application Information Retrieval (PAIR) system. Status information for published applications

may be obtained from either Private PAIR or Public PAIR. Status information for unpublished

applications is available through Private PAIR only. For more information about the PAIR

system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR

system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

/Q. C./

Examiner, Art Unit 2191

/Ted T. Vo/

Primary Examiner, Art Unit 2191